Trained K-Means Model

We have identified the optimal number of clusters is 4. Now, we run the algorithm again with 4 clusters:

```
#------------Train K-Means model with K = 4 (number of clusters = 4)-------------------
from sklearn.cluster import KMeans

# create a K-Means model that divides the data into 4 clusters.
kmeans = KMeans(n_clusters=4, init='k-means++', max_iter=300, n_init=10, random_state=42)
kmeans.fit(scaled_features)
print("Cluster Centers:", kmeans.cluster_centers_)
print("WCSS value of clusters:", kmeans.inertia_)
print("Number of iterations before reaching convergence:", kmeans.n_iter_)
print("Cluster assignment for each data point:", kmeans.labels_)
```

Output:

```
"D:\python\Customer churn rate analysis\.venv\Scripts\python.exe" "D:\python\Customer churn rate analysis\clustering.py"
Cluster Centers: [[ 1.06608015  0.9283784 ]
 [-0.88852937 -1.07477049]
 [-0.72731538  0.54124465]
 [ 0.87748746 -1.03948852]]
WCSS value of clusters: 2835.7163318393755
Number of iterations before reaching convergence: 8
Cluster assignment for each data point: [1 3 1 ... 1 2 0]
```

Save K-Means trained model as a file for submission

We will use **joblib** to save model. **joblib** is a Python library for saving and loading large models, especially models that involve large arrays.

```
import joblib
# Save the trained model to a file using joblib
joblib.dump(kmeans, filename: 'kmeans_model_joblib.pkl')
```

Output:

A file named “kmeans_model_joblib.pkl” is created inside current working directory. The file is ready to be submitted.

Load the model using joblib

To load the file above, we also use joblib:

```python
import joblib

# Load the trained model
loaded_model = joblib.load('kmeans_model_joblib.pkl')
# Use the loaded model for predictions
predictions = loaded_model.predict(scaled_features)
print("Cluster predictions:", predictions)
```

After successfully loaded the model, we can start applying methods and analyse the loaded model as normal. For example, we can:

- Predict cluster assignments for new data using **predict()**
  - In the screenshot above, we use **predict()** on the original data (scaled_features) to check the original data that was used during training, compare the cluster labels or validate the results
  - ```
    Cluster predictions: [1 3 1 ... 1 2 0]
    ```
  - The print output shows array that contains the cluster label for each data point in the dataset. This array is exactly the same as the array we printed previously (before exporting and loading the model). Hence, this proves that:
    - K_Means trained model before exported and loaded is the same as the K-Means trained model after exported and loaded => **kmeans_model_joblib.pkl** file contains the exact model that we want to export.
- Access the cluster centroids using **cluster_centers_**
- Evaluate the clustering quality by checking the inertia (WCSS) using **inertia_**
- Check the number of iterations it took for the model to converge using **n_iter_**
- Analyze distances of points to each centroid using **transform()**
    ```